AMENDMENTS TO THE CLAIMS

Please amend the claims as follows.

1. – 11. (Cancelled)

12. (Currently Amended) A computer-implemented method for verifying execution of a program comprising a plurality of code portions to be verified~~, wherein the program comprises a first code portion and a second code portion~~, the method comprising:

entering [[the]] a first code portion, wherein the first code portion comprises a first plurality of instructions beginning with a first entry point and ending with a first instruction that creates a first branch in the execution flow;

executing, by a processor, the first code portion;

calculating a first checksum during the execution of the first code portion, wherein the first checksum is calculated using ~~information~~ first operating code, wherein ~~associated with~~ at least one of the first plurality of instructions comprises the first operating code;

comparing the first checksum to a first pre-calculated checksum during execution of the first instruction that creates the first branch in the execution flow and prior to exiting the first code portion,

wherein the first pre-calculated checksum is passed as a parameter of the first instruction,

wherein the first pre-calculated checksum is calculated, during compilation of the program, using the first operating code, wherein the first operating code is generated during compilation of the program;

exiting the first code portion and entering the second code portion when the first checksum equals the first pre-calculated checksum, wherein the second code portion comprises a second plurality of instructions beginning with a second entry point and ending with a second instruction that creates a second branch in the execution flow; and

detecting an anomaly when the first checksum is not equal to the first pre-calculated checksum, wherein detection of the anomaly results in the second code portion remaining unexecuted.

13. (Currently Amended) The method of claim 12, further comprising:

~~entering the second code portion when the first checksum equals the first pre-calculated checksum, wherein the second code portion comprises a second plurality of instructions;~~

executing the second code portion;

calculating a second checksum during the execution of the second code portion, wherein the second checksum is calculated using second operating code, wherein ~~information associated with~~ at least one of the second plurality of instructions comprises the second operating code;

comparing the second checksum to a second pre-calculated checksum during execution of the second instruction that creates the second branch in the execution flow and prior to exiting the second code portion; and

exiting the second code portion if the second checksum equals the second pre-calculated checksum.

14. (Canceled)

15. (Currently Amended) The method of claim 12, wherein the first ~~a last~~ instruction in the first plurality of instructions to be executed prior to exiting the first code portion is modified to compare[[ing]] the first checksum to the first pre-calculated checksum.

16. (Previously Presented) The method of claim 12, wherein comparing the first checksum to the first pre-calculated checksum is performed after all of the first plurality of instructions have been executed.

17. (Canceled)

18. (Canceled)

19. (Cancelled)

20. (Currently Amended) A card comprising an electronic module, the electric module comprising:

a program, comprising a first code portion and a second code portion;

a first pre-calculated checksum~~, wherein the first pre-calculated checksum is calculated during compilation of the program~~;

a processor configured to execute the program, wherein executing the program comprises:

entering the first code portion, wherein the first code portion comprises a first plurality of instructions beginning with a first entry point and ending with a first instruction that creates a first branch in the execution flow;

executing the first code portion;

calculating a first checksum during the execution of first code portion, wherein the first checksum is calculated using ~~information~~ operating code, wherein ~~associated with~~ at least one of the first plurality of instructions comprises the operating code;

comparing the first checksum to the first pre-calculated checksum during execution of the first instruction that creates the branch in the execution flow and prior to exiting the first code portion, wherein the first pre-calculated checksum is passed as a parameter of the first instruction,

wherein the first pre-calculated checksum is calculated during compilation of the program using the operating code, wherein the operating code is generated during the compilation of the program;

exiting the first code portion and entering the second code portion when the first checksum equals the first pre-calculated checksum, wherein the second code portion comprises a second plurality of instructions beginning with a second entry point and ending with a second instruction that creates a second branch in the execution flow; and

detecting an anomaly when the first checksum is not equal to the first pre-calculated checksum, wherein detection of the anomaly results in the second code portion remaining unexecuted.

21. (Canceled)

22. (Currently Amended) A computer-implemented method for verifying execution of a program, ~~wherein the program comprises a first routine and a second routine,~~ the method comprising:

entering [[the]] a first routine, wherein the first routine comprises a plurality of instructions beginning with a first entry point and ending with a first instruction that creates a first branch in the execution flow, and each of the plurality of instructions is associated with a value;

initializing a counter associated with the first routine, prior to executing the first routine;

executing, by a processor, the first routine, wherein the counter is incremented by the value associated with each of the plurality of instructions executed during the execution of the first routine;

comparing a value of the counter to a pre-calculated value during execution of the first instruction that creates the branch in the execution flow and prior to exiting the first routine, wherein the pre-calculated value is passed as a parameter of the first instruction and wherein the pre-calculated value is calculated during compilation of operating code of the program associated with the at least one of the first plurality of instructions;

exiting the first routine and entering the second routine when the value of the counter equals the pre-calculated value, wherein the second routine comprises a second plurality of instructions beginning with a second entry point and ending with a second instruction that creates a second branch in the execution flow; and

detecting an anomaly when the value of the counter is not equal to the pre-calculated value, wherein detection of the anomaly results in the second routine remaining unexecuted,

wherein the first routine comprises [[a]] the first branch and a ~~second~~ third branch, and wherein the first branch and the ~~second~~ third branch are balanced to have the value of the counter resulting from executing instructions in the first branch equal to the value of the counter resulting from executing instructions in the ~~second~~ third branch.

23. – 24. (Canceled)

25. (Previously Presented) The method of claim 22, wherein the value associated with each of the plurality of instructions is unique.

26. (Previously Presented) The method of claim 22, wherein the value associated with a first one of the plurality of instructions is the same as the value associated with a second one of the plurality of instructions, if a type of the first one of the plurality of instructions is the same as a type of the second one of the plurality of instructions.

27. (Currently Amended) A card comprising an electronic module, the electronic module comprising:

a program[[,]] comprising a first routine and a second routine;

a value of a first pre-calculated counter;

a processor configured to execute the program, wherein executing the program comprises:

entering the first routine, wherein the first routine comprises a plurality of instructions beginning with a first entry point and ending with a first instruction that creates a first branch in the execution flow, and each of the plurality of instructions is associated with a value;

initializing a counter associated with the first routine, prior to executing the first routine;

executing the first routine, wherein the counter is incremented by the value associated with each of the plurality of instructions executed during the execution of the first routine;

comparing a value of the counter to a pre-calculated value during execution of the first instruction that creates the branch in the execution flow and prior to exiting the first routine, wherein the pre-calculated value is passed as a parameter of the first instruction and wherein the pre-calculated value is calculated during compilation of operating code of the program associated with the at least one of the first plurality of instructions;

exiting the first routine and entering the second routine when the value of the counter equals the pre-calculated value, wherein the second routine comprises a

second plurality of instructions beginning with a second entry point and ending with a second instruction that creates a second branch in the execution flow; and

detecting an anomaly when the value of the counter is not equal to the pre-calculated value, wherein detection of the anomaly results in the second routine remaining unexecuted,

wherein the first routine comprises a first branch and a third ~~second~~ branch, and wherein the first branch and the third ~~second~~ branch are balanced to have the value of the counter resulting from executing instructions in the first branch equal to the value of the counter resulting from executing instructions in the third ~~second~~ branch.

28. (Canceled)